Writeup Template

Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Rubric Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Histogram of Oriented Gradients (HOG)

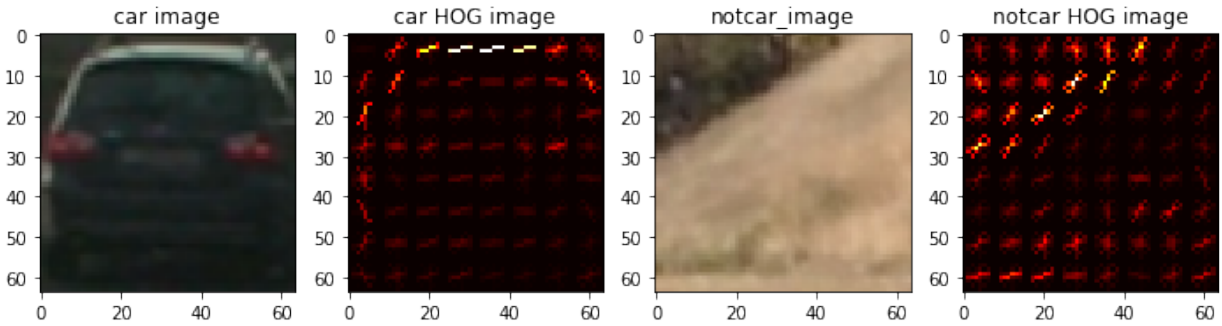
1. Explain how (and identify where in your code) you extracted HOG features from the training images.

The code for this step is contained in the first to 9th code cells of the IPython notebook called `vehicle-detection.ipynb`).

I started by reading in all the `vehicle` and `non-vehicle` images. Here is an example of one of each of the `vehicle` and `non-vehicle` classes:

I then explored different color spaces and different `skimage.hog()` parameters (`orientations`, `pixels_per_cell`, and `cells_per_block`). I grabbed random images from each of the two classes and displayed them to get a feel for what the `skimage.hog()` output looks like.

Here is an example using the `YCrCb` color space and HOG parameters of `orient=9`, `orientations=8`, `pixels_per_cell=(8, 8)` and `cells_per_block=(2, 2)`:



2. Explain how you settled on your final choice of HOG parameters.

I tried various combinations of parameters and here is the result.

	1	2	3	4	5	6	7	8
color_space	YCrCb	LUV, YUV	HLS					
orient	9			11				
pixels_per_cell	8				16			
cells_per_block	2					4		
hog_channel	“ALL"						0	
spatial_size	(32,32)							(16,1
hist_bins	32							
spatial_feat	TRUE							
hist_feat	TRUE							
hog_feat	TRUE							
feature vector length	8460		8460	9636	4140	13968	4932	6156
Time for computing features	61.14	Error	61.98	63.23	49.39	50.86	30.02	59.2
n_samples	3000							
Accuracy	0.9904		0.9921	0.991	0.9834	0.9916	0.9747	0.99
Time for training	23.97	Error	30.04	13.37	7.77	34.43	11.21	12.9

If you want to reduce the time for computing features, then you should reduce pixels_per_cesll, cells_per_block or set hog channel as 0. Reducing pixels_per_cell or setting hog_challel to 0 decrease the length of the feature vector, so you can save time for taining too.

I also reviewed the result of it on the piplined video, then I select the parameters on the first column. It produce the least vehicle windows.

3. Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

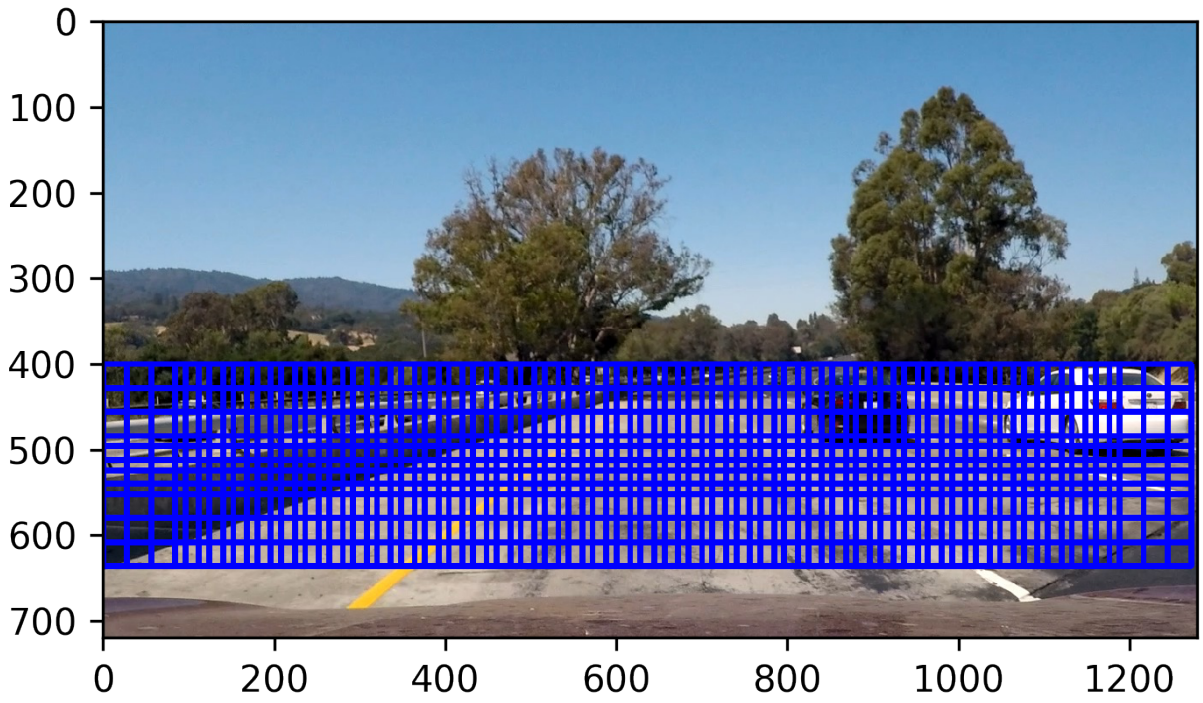
I trained a linear SVM using the binned color features, the color histogram features and hog features(in th third and fourth code cells). Before training it, I normalized the concatenated all the features with StandardScaler().fit(). In addition, I divided them into a training group and a test group(in th ninth and fourth code cells).

Sliding Window Search

1. Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

The code for this step is contained in the tenth code cell of the IPython notebook, `vehicle-detection.ipynb`. I decided to search window positions from the middle of the image because there are only the sky, trees and static things. The start position of y is 400 and cut the bottom part. It's just the bonnet of a car.

The car ahead of me is usually smaller then 100x100, so the window size is 96x96. It works well on the video. A small window like 64x64 cannot detect a car next to my car. A big window like 128x128 looks good, but it is bigger than the car in the video.



2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

Ultimately I searched on 96x96 scale using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result. Here are some example images:

Video Implementation

1. Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

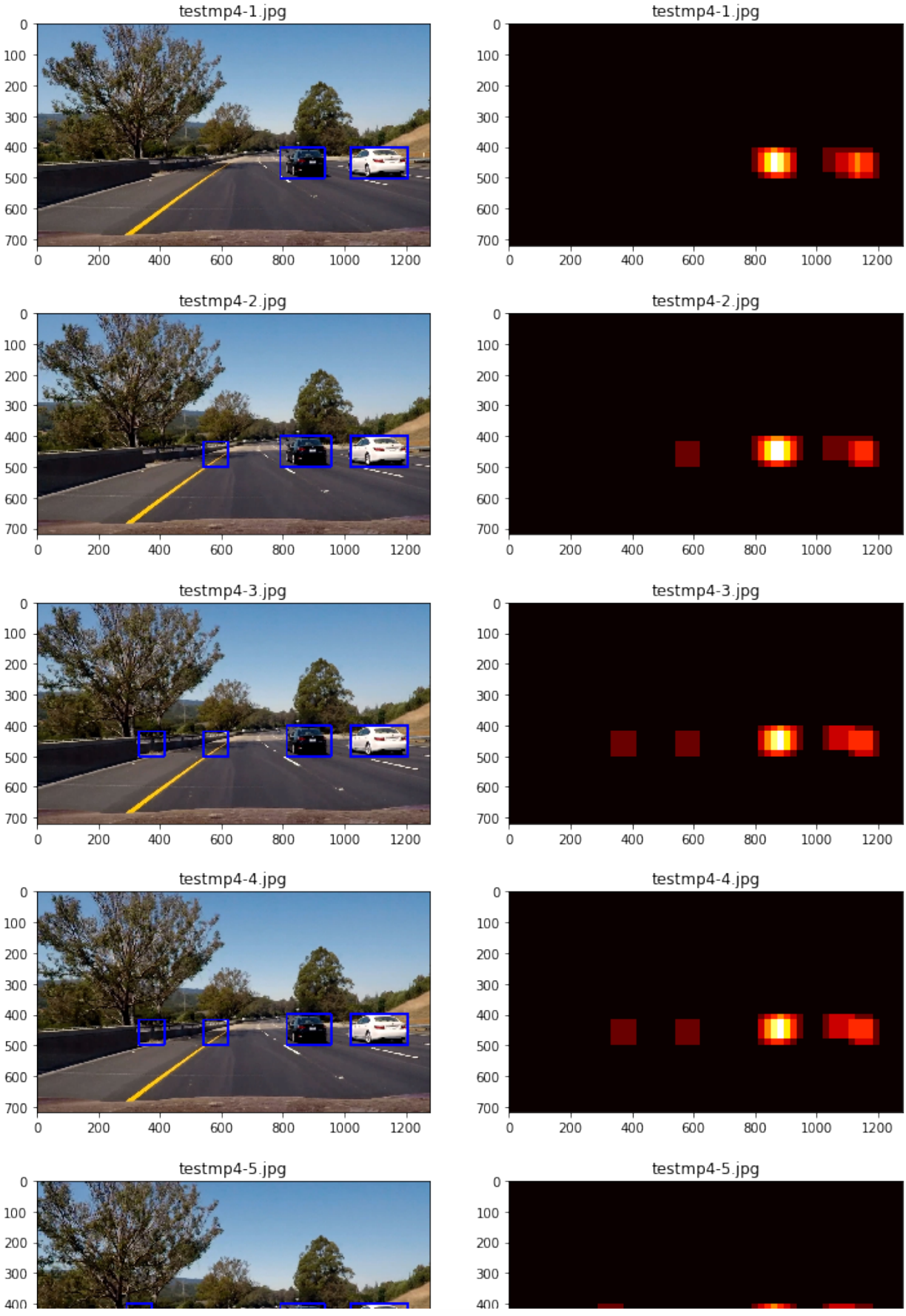
Here's a link to my video result

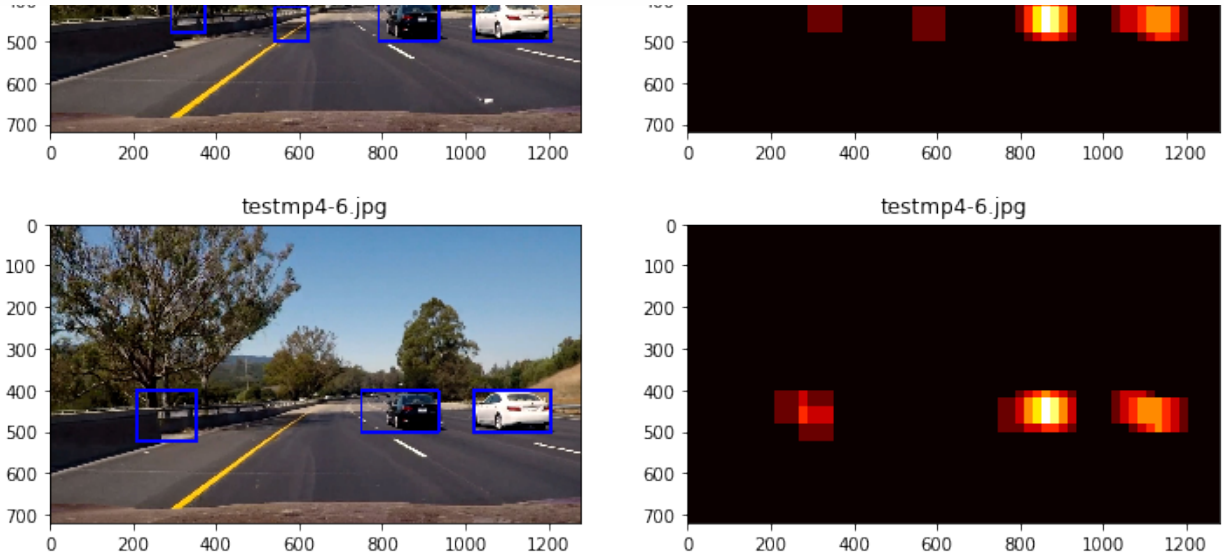
2. Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

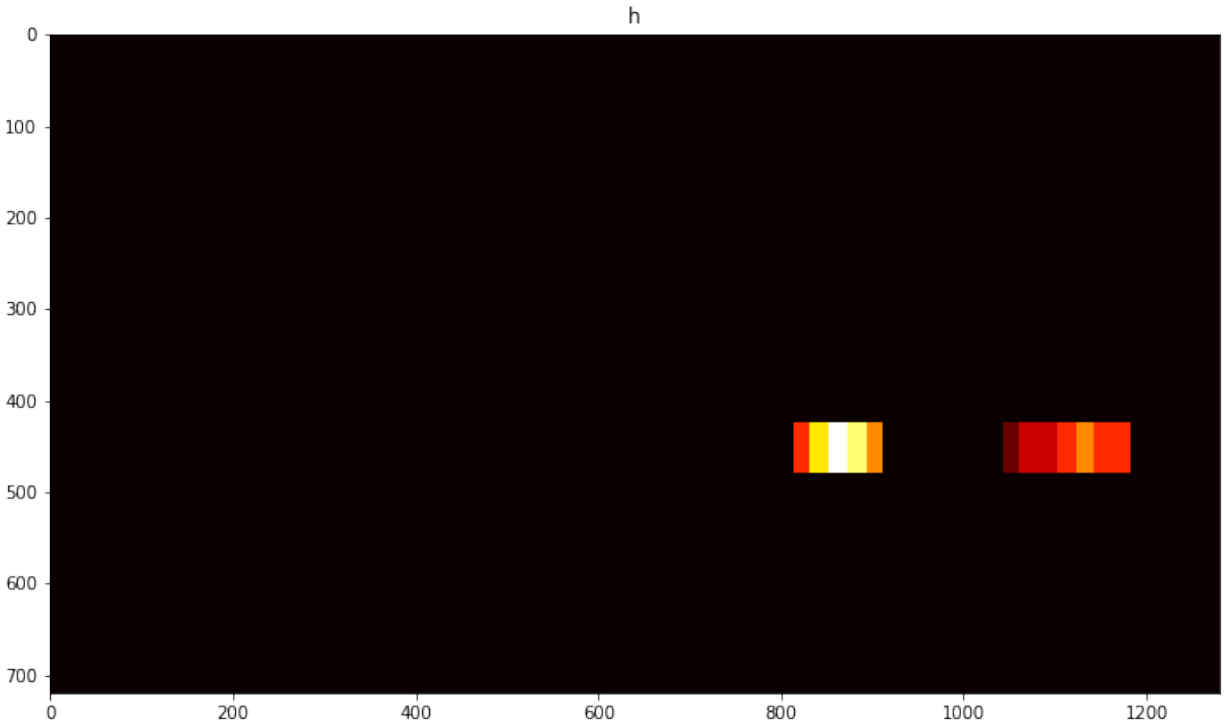
Here's an example result showing the heatmap from a series of frames of video, the result of `scipy.ndimage.measurements.label()` and the bounding boxes then overlaid on the last frame of video:

Here are six frames and their corresponding heatmaps:





Here is the output of `scipy.ndimage.measurements.label()` on the integrated heatmap from all six frames:



Here the resulting bounding boxes are drawn onto the last frame in the series:

Discussion

1. Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

Here I'll talk about the approach I took and what techniques I used:

1. Read image files
    - glob(): finds all the pathnames matching a specified pattern
2. Extract HOG features
    - histogram(): compute the histogram of a set of data
    - ravel(): return a contiguous flattened array
    - hog(): compute a Histogram of Oriented Gradients (HOG)
3. Train them
    - StandardScaler(): standardize features by removing the mean and scaling to unit variance
        - fit(): compute the mean and std to be used for later scaling
        - transform(): perform standardization by centering and scaling
    - train_test_split(): split arrays or matrices into random train and test subsets
    - LinearSVC(): Linear Support Vector Classification.
        - fit(): fit the model according to the given training data.
4. Search for vehicles
    - sliding window: start from the left-middle to right-bottom of the image, slide by 70% of the window

    - LinearSVC()
        - predict(): predict class labels for samples in X.
5. Create heatmap
    - label(): label features in an array
6. Draw a box for a vehicle
    - tracking windows: insert and keep the last position of vehicle, the number of detections, etc
7. Delete the false detection
    - threshold: display windows which appear more than 20 times and delete which is not used recently
8. Make it smooth
    - average: aeverage the last 10 windows' positions
9. Pipeline on a video stream: VideoFileClip

Although the accuracy of prediction of image classification is 99%, the false positive detection window appears a lot. So, I delete all of them by increasing the minimum number of consecutive times from 6 number to 20. It woks well now, but the first detection of a vehicle is slow than before.

I'll use the different sizes of sliding window and detection window according to the y position because a vehicle size is small as they are farther. The multi-windows might detect various vehicles like a truck or a trailer which cannot be included in a small window.